**ExtraExDS\_4Trees – Exercise 2**

**Specification**

class binary\_tree[node]

genres b\_tree, node, label, int, bool

operations

parent: node b\_tree 🡪 node

left\_child: node b\_tree 🡪 node

right\_child: node b\_tree 🡪 node

label: node b\_tree 🡪 label

create: b\_tree b\_tree label -> b\_tree

root: b\_tree 🡪 node

makenull: b\_tree 🡪 b\_tree

length: b\_tree -> int

same\_num\_nodes: b\_tree btree 🡪 bool

compare\_form: b\_tree b\_tree 🡪 bool

same\_form: b\_tree b\_tree 🡪 bool

sum\_nodes: b\_tree b\_tree 🡪 b\_tree

sum\_btrees: b\_tree b\_tree 🡪 b\_tree

endspec

**Implementation**

node = record

element : label

leftchild: ^node

rightchild: ^node

parent: ^node

endrecord

label: elementtype

class B\_Tree

private root: ^node

public ^node parent(n: node)

public ^node left\_child(n: node)

public ^node right\_child(n: node)

public label label(n: node)

public void create(leftTree, rightTree: ^node, l: label)

public ^node root()

public void makenull()

public int num\_nodes(n: ^node)

public bool same\_num\_nodes(n1, n2: ^node)

public bool compare\_form(n1, n2: ^node)

public bool same\_form(n1, n2: ^node)

public ^node sum\_nodes(n1, n2: ^node)

public ^node sum\_btrees(n1, n2: ^node)

endclass

public bool B\_Tree :: num\_nodes(n: ^node)

counter: int

counter := 0

if ( n == null)

return counter

else

counter += 1

counter += num\_nodes(n^.leftchild)

counter += num\_nodes(n^.rightchild)

return counter

endif

endmethod

public bool B\_Tree :: same\_num\_nodes(n1, n2: ^node)

num1, num2 : int

num1:= num\_nodes(n1)

num2:= num\_nodes(n2)

if (num1 == num2)

return 1

else

return 0

endif

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the first tree and from the second tree to get the total number of nodes and compare them to know if they have the same number of nodes.

public bool B\_Tree :: compare\_form(n1, n2: ^node)

verify : bool

verify := 1

if ( n1 == null && n2 == null)

return 1

else if (( n1 != null && n2 == null) || ( n1 == null && n2 != null))

return 0

else

if (n1^.rightchild == null && n1^.leftchild == null)

if(n2^.rightchild == null && n2^.leftchild == null)

return 1

else

return 0

endif

else if (n1^.rightchild == null)

if(n2^.rightchild == null && n2^.leftchild != null)

return compare\_form(n1^.leftchild, n2^.leftchild)

endif

return 0

else if (n1^.leftchild == null)

if(n2^.rightchild != null && n2^.leftchild == null)

return compare\_form(n1^.rightchild, n2^.rightchild)

endif

return 0

else

if(n2^.rightchild != null && n2^.leftchild != null)

verify := compare\_form(n1^.leftchild, n2^.leftchild)

if(verify)

return compare\_form(n1^.rightchild, n2^.rightchild)

endif

return verify

endif

return 0

endif

endif

endmethod

public bool B\_Tree :: same\_form(n1, n2: ^node)

verify: bool

verify:= same\_num\_nodes(n1, n2)

if (!verify)

return 0

verify := compare\_form(n1, n2)

return verify

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the first tree and from the second tree to get the total number of nodes and compare them to know if they have the same number of nodes and if it they have the same number of nodes, then they structures are compare to see if they have the same structure, even though their labels are not the same.

public ^node B\_Tree :: sum\_nodes(n1, n2: ^node)

temp : ^node

allocate (temp)

if(n1 == null) {We don’t need to indicate anything as they have the

return null same structure}

endif

temp^.element := n1^.element + n2^.element

temp^.rightchild := sum\_nodes(n1^.rightchild, n2^.rightchild)

temp^.leftchild := sum\_nodes(n1^.leftchild, n2^.leftchild)

return temp

endmethod

public ^node B\_Tree :: sum\_btrees(n1, n2: ^node)

if( compare\_form(n1, n2))

temp: ^node

return sum\_nodes(n1, n2: ^node)

return null

endmethod

Running Time: O(n) 🡪 We need to go through all the elements from the first tree and from the second tree in order to put their added label in the specific node from a new tree and return it once we know they have the same structure and even if they had different structure, the running time would be the same.